LZ SQUEEZER
A Compression Technique based on LZ77 and LZ78

Govind Prasad Arya*, Arvind Singh**, Rahul Painuly***, Shashank Bhadri**** & Sunakshi Maurya*****

*Assistant Professor & Research Guide, Department of Computer Science, Shivalik College of Engineering, Dehradun, Uttarakhand, INDIA.
E-Mail: govind.arya10@gmail.com
**Research Scholar, Department of Computer Science, Shivalik College of Engineering, Dehradun, Uttarakhand, INDIA.
E-Mail: arvindattechies@gmail.com
***Research Scholar, Department of Computer Science, Shivalik College of Engineering, Dehradun, Uttarakhand, INDIA.
E-Mail: rahulpainulyS@gmail.com
****Research Scholar, Department of Computer Science, Shivalik College of Engineering, Dehradun, Uttarakhand, INDIA.
E-Mail: shashankbhadri36@gmail.com
******Research Scholar, Department of Computer Science, Shivalik College of Engineering, Dehradun, Uttarakhand, INDIA.
E-Mail: sunakshi.maurya@facebook.com

***Abstract*—**In present era internet is the most prominent media for communication, and we transmit most of our data through this medium. Since there is a trade of between efficiency of internet and size of the data being transmitted, so less size data can be reached fast and efficient. Therefore, compressing the data provide a significant change in efficiency of the transmission of data. And on other side most of the date which is stored on storage disk has some statistical redundancy, which consume more unnecessary space and it can be solved by compressing the data.

***Keywords*—**Compression, Decompression, Dictionary, LookAheadBuffer, SearchBuffer, Symbol, Temporary File, Tuple

I. INTRODUCTION

COMPRESSION is the reduction in size of data in order to save space and transmission time [Arnavut, 2000]. Compression is of two types 1- lossless compression [Burrows & Wheeler, 1994; Awan & Mukherjee, 2001], and 2- lossy compression. In lossy compression data once being compressed can't get back to its original form e.g. video compression, while in lossless compression the compressed data can be get back to its original form e.g. text compression. In lossy compression if we to reduce the size of a video file by reducing its pixel then it can't be get back to its original form. On the other side when we compress a text file it does not have any significance if it can't get back to its original form. There are many lossless compression algorithm which we studied e.g. LZ77, LZW, Huffman compression [Balkenhol et al., 1999] and did overcome some problems of these algorithm, which we are discussing in our algorithm.

1.1. LZ-77 Algorithm

```
While (lookAheadBuffer not empty)
{
    get a reference (position, length) to longer match from
    search buffer;
        if (length>0)
        {
        Output (position, length, next symbol);
        Shift the window length+1 position along;
        }
        else
        {
        Output (0, 0, first symbol in lookAheadBuffer);
        Shift the window 1 position along;
        }
}
```

Search Buffer: It is the storage structure in which characters are searched for match and it is initially empty.

LookAheadBuffer: It is also the storage for character from which the characters are matched into search buffer.

Tuple: It is combination of three character as (x,y,z), where x is the length where first character from

lookAheadBuffer matched into Search Buffer, and y is the offset i.e. the number of character from lookAheadBuffer which matched in the Search Buffer.

1.2. Shortcomings of LZ77 Algorithm

- In their original LZ77 algorithm; Lampel and Ziv proposed that all string be encoded as a length and offset, even string with no match.
- In LZ77, search buffer is thousands of bytes long, while the lookahead buffer is tens of byte long.
- The encryption can be much time consuming due to the larger number of comparison need to be made to find matched pattern.
- LZ77 doesn't have its external dictionary which cause problem while decompressing on another machine.

In this algorithm whenever there is no match of any strings it encoded that string as a length and offset which will take more space and this unnecessary step also increases time taken by the algorithm.

It also have thousands of bytes long search buffer, when we check for any character in our string it will go through all the thousands of bytes in our search buffer which is not time consuming at all, so we advanced our algorithm little bit so that it will just search for some limited no of bytes so that it doesn't have to go for thousands of bytes it simply match the character in a search buffer by lookahead and replace it with any symbol if no match found then it simply replaces it by itself that's why it is very time consuming.

II. METHODOLOGY

2.1. Our Contribution

Unlike the LZ-77, the proposed algorithm does not generate new output tuple when no match found, rather it replaces the symbol by itself in tuple column. And uses an external dictionary contrary to LZ-77 (generate internal dictionary while encoding the text). Which remove problem of LZ-77 & 78 i.e. dictionary [Franceschini & Mukherjeem, 1996] grows without bound.

In this algorithm we have a window of array like LZ-77 in which characters are inserted to find match, and window is divided into two segments: searchBuffer and lookAheadBuffer. But unlike LZ-77 in this search buffer size is very limited i.e.11 byte and lookAheadBuffer size is 5 byte, which removes shortcoming of previous algorithm, i.e. it take lesser time to compare to match than LZ-77.

2.2. Compression Algorithm

```
while (source input not empty)
 {
      Insert character into lookahead buffer;
        while (lookaheadbuffer not empty)
        {
         Get a reference (position, length) to longest match;
            if (length>0)
             {
                Output (position, length, root symbol);
                 Shift the window length+1, position along ;
                 Replace output tuple with its corresponding
symbol in external dictionary
             }
            else
             {
              Insert character first character into output file
              Shift window one position along
             }
        }
 }
```

We can get a compressed file after applying above algorithm.

2.3. Compression and Decompression Process

In this section we show how the compression and decompression is performed using proposed algorithm.

2.3.1. Compression Process

Input Data: abbcababcbb

Here in this example input stream is the data which has to be compress and it has a size of 11 byte and compression need to be done to make its size less than 11 bytes. Here first characters are inserted in to lookahead buffer similar to LZ77 if a match is found in search buffer then it generate output tuple otherwise write the character itself in temporary file. For example initially when character 'a' is inserted into lookAheadBuffer then program look into search buffer [Balkenhol et al., 1999, Chapin & Tate, 1998] to find the match but it does not find any match, since initially search buffer is empty therefore it generate the character 'a' itself in output tuple and get shifted to search buffer. Similarly it get processed for first occurrence of character 'b', but when second 'b ' is inserted into lookAheadbuffer and it find a match into search buffer i.e. character 'b', so it generate output tuple as (1,1,c). Where first '1' shows one index before character get matched and second '1' shows that only one character is matched and character 'c' shows that it is the character which does not get matched. Similarly this process is repeated for all character and we find output tuple. Which are lesser in number than character into input data as in the example taken output tuple are 6. Then these output tuple are stored into a temporary or intermediate file.

Table 1 – Compression Process

← Search Buffer →																lookahead buffer	Output Tuple
-	-	-	-	-	-	-	-	-	-	-	a	b	b	c	a	babcbb	a
-	-	-	-	-	-	-	-	-	-	a	b	b	c	a	b	abcbb	b
-	-	-	-	-	-	-	-	-	a	b	b	c	a	b	a	bcbb	(1,1,c)
-	-	-	-	-	-	-	a	b	b	c	a	b	a	b	c	Bb	(4,2,a)
-	-	-	-	a	b	b	c	a	b	a	b	c	b	b	-		(5,2,b)
-	A	b	b	c	a	b	a	b	c	b	b	-	-	-	-		(1,1, empty)
a	B	b	c	a	b	a	b	c	b	b	-	-	-	-	-		

Temporary Data: ab (1, 1, c)(4, 2, a) (5, 2, b) (1, 1, EOF)

Now one of the major contribution which we have done in this compression technique i.e. we provide an external dictionary in which we have symbol (non keyboard symbol) corresponding to each tuple. As shown in the table external dictionary. Now temporary file is compared with external dictionary, and tuples are replaced with its corresponding symbol to generate output file. As in our example we get output file as our compressed file which have data: - ab♥ ■▬▪

Now we can see compressed file have only 6 character rather its 11 character in original input.

Compression Ratio = ((Original Size – Compressed Size) / Original Size)*100
= ((11-6)/11)*100=45.45%

Table 2 – External Dictionary

Tuple	Symbol
(1,1,a)	→
(2,1,a)	↔
(4,2,a)	■
(5,2,b)	▬
(1,1,c)	♥
(1,1,d)	●
(1,1,EOF)	▪

2.3.2. Decompression Process

In this process the compressed code is get back to its original form.

Here is the compressed data: ab♥ ■▬▪

Now we have to decompress [Cleary, 1995 & Effros 2000] this compressed data into original data. For every symbol we have its corresponding tuple in its external dictionary [Burrows & Wheeler, 1994] and each symbol is replaced with its corresponding tuple, the output now is temporary data in which we operate to get original data. As in temporary data [Balkenhol & Shtarkov, 1999] first of all we get character 'a', since it is a single character so it get replaced by itself in decompress file. Similarly this repeated for character 'b'. When tuple (1, 1, c) occur first '1' shows that one character before current position is matched character i.e. character 'b'. And second '1' shows only one match is found for this tuple while 'c' shows it is the symbol which did not get matched. In this way the original data is obtained from temporary data.

Table 3 – Decompression Process

Symbols	Corresponding Tuples
a	a
b	b
♥	(1,1,c)
■	(4,2,a)
▬	(5,2,b)
▪	(1,1,EOF)

Temporary Data: ab (1, 1, c) (4, 2, a) (5, 2, b) b

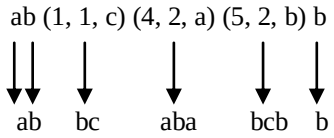

Figure 1 – Generation of Original Data

Now we can see that we have got the data which we had before compression.

2.3.3. Advantages of Algorithm

- This algorithm does not generate new tuple when no match found rather it replaces the character by itself.
- Search buffer size in this algorithm is relatively very small so it is not time consuming and is faster than LZ77.
- It has its external dictionary which make it easy to decompress while comparing tuple with external dictionary

III. CONCLUSION & RESULT

The main aim of compression is to reduce the size of the data so that it will take less space and easily transfer over the internet which reduces the cost. LZ-77 is a good algorithm for compression but still it has some major drawbacks as it take much time to compress while comparing for pattern match, while in our algorithm by adding external dictionary and limited search buffer size we overcome this problem and which make it faster than previous algorithm. As for example taken to describe the process the compression ratio is 45.45%. and the future scope of algorithm can be improved more are by providing threading in program to make program run concurrently [Chapin, 2001] and taking appropriate data structure so dictionary can be accessed fast.

REFERENCES

[1] M. Burrows & D.J. Wheeler (1994), "A Block – Sorting Lossless Data compression Algorithm", *SRC Research Report 124*, Digital Research Systems Research Centre.

[2] J.G. Cleary, W.J. Teahan & Ian H. Witten (1995), "Unbounded Length Contexts for PPM", *Proceedings of Data Compression Conference, IEEE Computer Society*, Snowbird Utah, Pp. 52–61.

[3] R. Franceschini & A. Mukherjee (1996), "Data Compression using Encrypted Text", *IEEE Proceedings of ADL*, Pp. 130–138.

[4] B. Chapin & S. Tate (1998), "Preprocessing Text to Improve Compression Ratios", *Proceedings of the IEEE Data Compression Conference*, Snowbird, Pp. 532.

[5] B. Balkenhol, S. Kurtz & Y.M. Shtarkov (1999), "Modifications of the Burrows Wheeler Data Compression Algorithm", *Proceedings of Data Compression Conference, IEEE Computer Society*, Snowbird Utah, Pp. 188–197.

[6] B. Balkenhol & Y. Shtarkov (1999), "One Attempt of a Compression Algorithm using the BWT", *SFB343: Discrete Structures in Mathematics, Faculty of Mathematics, University of Bielefeld*, Germany.

[7] Z. Arnavut (2000), "Move-to-Front and Inversion Coding", *Proceedings of Data Compression Conference, IEEE Computer Society*, Snowbird, Utah, Pp. 193–202.

[8] M. Effros (2000), "PPM Performance with BWT Complexity: A New Method for Lossless Data Compression", *Proceedings of Data Compression Conference, IEEE Computer Society*, Snowbird Utah, Pp. 203–212.

[9] F.S. Awan & A. Mukherjee (2001), "LIPT: A Lossless Text Transform to Improve Compression", *Proceedings of International Conference on Information Technology: Coding and Computing*, Pp. 452–460.

[10] B. Chapin (2001), "Higher Compression from the Burrows–Wheeler Transform with New Algorithms for the List Update Problem", Ph.D. Dissertation, *Department of Computer Science, University of North Texas*.

Govind Prasad Arya has completed Master of Technology (M.Tech) in Computer Science & Engineering from Uttarakhand Technical University, Dehradun (Uttarakhand). He has 8 years of teaching experience from different institutions (Govt. Polytechnic Dwarahat, Kumaon Engineering College Dwarahat). Presently He is working at Shivalik College of Engineering, Dehradun as an Assistant Professor in Computer Science & Engineering Department. Data Structures, Computer Organization, Operating System & Compiler Design are his area of interest. His four research papers on Compression have been published by reputed International Journals. He has attended 4 Conferences and 6 Seminars.

Arvind Singh, pursuing his B. Tech, Final year in Computer Science & Engineering. From Shivalik College of Engineering. His area of interest is data optimization, Cloud Computing, Data mining. for his College project. He has attended several seminar and workshop on cloud computing, Data mining & Data warehousing conducted by Microsoft, Oracle, IBM etc. He is also interested in software development and have developed FumbleWeb (an open source web browser), and data compressor.

Rahul Painuly, pursuing his B. Tech, Final year in Computer Science & Engineering. From Shivalik College of Engineering. His area of interest is Software Designing, Cloud Computing and Networking, Ethical Hacking. He has attended several seminar and workshop on cloud computing, Networking, Ethical Hacking conducted by Microsoft, Oracle, IBM, Ankit Fadia respectively.

Shashank Bhadri, pursuing his B. Tech, Final year in Computer Science & Engineering. From Shivalik College of Engineering. His area of interest is Social Networking. He has attended several seminar and workshop on cloud computing, Web Designing, Embedded System conducted by Microsoft, IBM, Sofcon Enterprise etc. He is highly devoted for wild life research.

Sunakshi Maurya, pursuing her B. Tech, Final year in Computer Science & Engineering. From Shivalik College of Engineering. Her area of interest is Developer, Cloud Computing and Web Designing. She has attended several seminar and workshop on cloud computing, Web Designing, Ethical Hacking conducted by Microsoft, PicNFrames, Ankit Fadia Certification respectively.